National Défense
Defence nationale

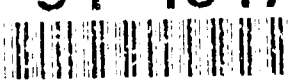
DEVELOPMENT ENVIRONMENT FOR DIINS (Dual Inertial Integrated Navigation System)

by

J.C. McMillan and R. Ramotaur

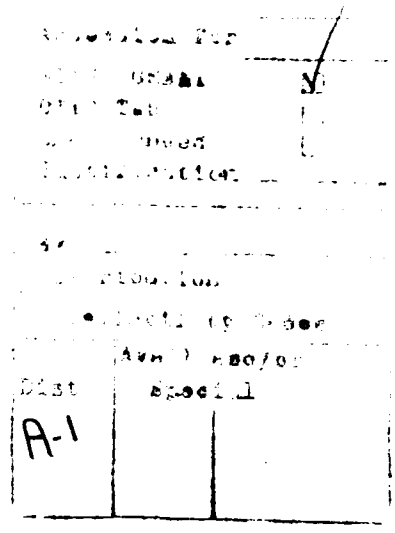
DEFENCE RESEARCH ESTABLISHMENT OTTAWA
TECHNICAL NOTE 91-5

Canada



April 1991
Ottawa

National Défense
Defence nationale



DEVELOPMENT ENVIRONMENT FOR DIINS
(Dual Inertial Integrated Navigation System)

by

J.C. McMillan and R. Ramotaur
Navigation and Integrated Systems Section
Electronics Division

DEFENCE RESEARCH ESTABLISHMENT OTTAWA
TECHNICAL NOTE 91-5

PCN
041LJ

April 1991
Ottawa

ABSTRACT

With a view to satisfying the navigational requirements of TRUMP, CPF and future submarines, DREO initiated an investigation into the possible extension of MINS (Marine Integrated Navigation System) to accommodate the two inertial navigation systems which each of these platforms were expected to have, in addition to the various navigation sensors that MINS already integrates (GPS, Transit, Loran-C, Omega, Speedlog and Gyrocompass). It soon became clear that the ideal solution would be a new system with a different architecture than MINS. The new concept was called DIINS (Dual Inertial Integrated Navigation System). A study was therefore initiated to determine the most appropriate hardware and software to be used in the design and development of DIINS. This report contains a brief description of the recommended hardware and software solution, as well as the rationale for their selection.

RÉSUMÉ

Dans le but de satisfaire les besoins de navigation de TRUMP, de CPF et de futurs sous-marins, DREO a entrepis une enquête sur une extension possible de MINS (système intégré de navigation de la Marine). Cette extension aurait pour but d'accomoder les deux systèmes de navigation par inertie qui devaient faire pârtie de chacun des projets ci-dessus, en plus des différent détecteurs de navigation que MINS englobe déjà (GPS, Transit, Loran-C, Omega, Speedlog et Gyrcompass). Il est vite devenu évident que la solution idéale consisterait en un nouveau système avec une architecture différente de MINS. Le nouveau concept fut appellé DIINS (double système intégré de navigation par inertie). Une étude a donc été entreprise pour déterminer le matériel et le logiciel les plus appropriés pour être utilisés dans la conception et le développement de DIINS. Ce rapport contient une brève description de la solution recommendée pour le matériel et le logiciel, ainsi qu'une analyse raisonnée de leur sélection.

EXECUTIVE SUMMARY

With a view to satisfying the navigational requirements of TRUMP, CPF and future submarines, DREO initiated an investigation into the possible extension of MINS (Marine Integrated Navigation System, described in reference [1]) to accommodate the two inertial navigation systems which each of these platforms were expected to have, in addition to the various navigation sensors that MINS already integrates (GPS, Transit, Loran-C, Omega, Speedlog and Gyrocompass). It soon became clear that the ideal solution would be a new system with a different architecture than MINS. The new concept was called DIINS (Dual Inertial Integrated Navigation System).

In September 1988, DREO initiated a study to determine the hardware and software appropriate to the needs of the DIINS application. This report, contains a brief description of the recommended hardware/software solution, as well as the rationale for the decision.

After a detailed study of the requirements and careful analysis of the alternative options and systems the conclusions were the following:

1. The Ada Programming Language is the most suitable implementation language,
2. The Telesoft Ada compiler is the best production quality compiler on the market,
3. A suitable real-time executive must be acquired to enhance the real-time capabilities of the Ada runtime system. It must be appropriate to the needs of the target configuration,
4. The Sun 3/60 (Sun 3/80) workstations provide the most appropriate price/performance capabilities as a host system,
5. The Motorola 680X0 family of microprocessors are the most suitable target, and
6. The VME bus architecture is a suitable bus architecture, based on its popularity, compatibility and performance.

It is therefore recommended that DREO acquire:

- three Sun 3/60 (Sun 3/80) workstations (including 800 megabyte hard disk, Sun O/S and accessories) and
- RTAda Comprehensive Development System.

TABLE OF CONTENTS

ABSTRACT iii
EXECUTIVE SUMMARY v
1 INTRODUCTION 1
1.1 Objective 1
1.2 Background 1
1.3 Summary Recommendations 2
2 SOFTWARE REQUIREMENTS 5
2.1 Large Embedded System 5
2.2 High-Level Language 5
2.3 Fault-Tolerant 5
2.4 Rate of Input 5
2.5 Processing Time 5
2.6 Support for Concurrency 6
2.7 Support for Data Abstraction 6
2.8 Support for Real-Time 6
3 COMPARISON OF PROGRAMMING LANGUAGES 7
3.1 Large Embedded System 7
3.1.1 Maintenance 7
3.1.2 System Decomposition 8
3.2 Fault-Tolerant 8
3.3 Support for Concurrency 9
3.4 Support for Data Abstraction 9
3.5 Support for Real-Time 9
3.6 Programmer Availability 9
4 PROGRAMMING SUPPORT ENVIRONMENT 10
4.1 Compiler 10
4.2 Global Optimizer 13
4.3 Library Manager 13
4.4 Source Level Debugger 13
4.5 Host Profiler 13
4.6 Downloader and Receiver 14

5 HOST SYSTEMS 15
5.1 Costs 15
5.2 Performance 15
5.3 Compatibility with Target 15
5.4 User Friendly 16
5.5 VME 16
6 TARGET SYSTEMS 17
6.1 Microprocessors 17
6.1.1 iAPX80386 17
6.1.2 iAPX80960 17
6.1.3 TMS320 18
6.1.4 MC680X0 18
6.2 Real-Time Executives 18
6.2.1 Problems of Ada Runtime System 18
6.2.2 Ada Real-Time Executive (ARTX) 20
6.3 Bus Architectures 21
6.3.1 Synchronous Versus Asynchronous 22
6.3.2 Data/Address Bus 22
6.3.3 Arbitration Scheme 22
6.3.4 Interrupt Handling 23
6.3.5 Processor Independence 23
6.3.6 Multiple Vendors 23
7 RESULTS 24
REFERENCES 25

1 INTRODUCTION

1.1 Objective

With a view to satisfying the navigational requirements of TRUMP, CPF and future submarines, DREO initiated an investigation into the possible extension of MINS (Marine Integrated Navigation System, described in reference [1]) to accommodate the two inertial navigation systems which each of these platforms were expected to have, in addition to the various navigation sensors that MINS already integrates (GPS, Transit, Loran-C, Omega, Speedlog and Gyrocompass). It soon became clear that the ideal solution would be a new system with a different architecture than MINS. The new concept was called DIINS (Dual Inertial Integrated Navigation System).

Since the DIINS software was expected to be significantly larger and more complex than the MINS software, and several people were expected to work in parallel on different aspects of DIINS, it was suspected that a different programming language and a new software development environment would be required. Therefore the Defence Research Establishment Ottawa initiated a study to determine the hardware and software appropriate to the needs of the DIINS application. This report provides a brief description of the scale of the DIINS project from a software point of view. It contains a brief description of the recommended hardware/software development environment, as well as the rationale for the recommendation. It also describes the software development environment which was eventually acquired for DIINS work.

1.2 Background

As described in reference [1] the MINS software consists of 27 tasks with a total of about 25,000 lines of code and is written in the C language. This software was largely developed on a VAX 11/780 computer in the early to mid 1980's. The MINS production model uses a Motorola 68020/68881 processor running at 10 MHz and is essentially running at capacity with the MINS software.

The processing burden for DIINS was expected to be significantly greater than for MINS for the following reason. The processing burden for MINS was to a large extent dominated by the covariance propagation subroutine in the Kalman filter of the "Navigation/Filter" task (see reference [1]), especially before the introduction of the more efficient Bierman-Agee-Turner covariance propagation method. The computational burden imposed by covariance propagation is determined by the dimension of the state vector and the propagation rate. Generally the computation requirement for covariance propagation increases as the square of the state dimension and increases linearly with the frequency of propagation. The production model of MINS employs a 17 state Kalman filter, for which the covariance matrix is propagated every 20 seconds. Preliminary design options for DIINS suggest that several (up to five) filters with between 12 and 30 states each, will be required to run in parallel, at a frequency of at least every 10 seconds (preferably every second). Thus there will be at least an order of magnitude

increase in the computational burden in DIINS as compared to MINS, and perhaps two orders of magnitude.

The MINS production model uses a Motorola 68020/68881 processor running at 10 MHz and is essentially running at capacity with the MINS software. In fact a better comparison can be made with the preproduction model of MINS, which did not employ the more efficient Bierman-Agee-Turner routine, since this method cannot be used in DIINS (because it requires certain Kalman filter design matrices to have an especially simple form which is not the case with DIINS). The preproduction MINS, with the same 68020/68881 processor running at 10 MHz, was running at capacity with only 15 states and a 30 second propagation rate.

Thus five 20 state filters running at one Hz requires $5 \times (20/15)^2 \times 30 = 267$ times more computational capacity than one 15 state filter running at 1/30 Hz using the same algorithm. A more realistic projection of DIINS computational requirement would be four filters with 12, 14, 15 and 18 states respectively, running at different rates of from one to ten seconds. The increased burden in comparison to MINS then becomes: $(12/15)^2 \times 30 + (14/15)^2 \times 30 + (15/15)^2 \times 3 + (18/15)^2 \times 3 = 19.2 + 26.1 + 3 + 4.3 = 52.6$ This will likely be quite manageable by available processors by the time the DIINS design and simulation analysis phase is completed.

The increased complexity of DIINS, in comparison to MINS, is due in part to the fact that these different filters must be coordinated to share the same measurements and FDIR (Failure Detection Isolation and Reconfiguration) information. The FDIR itself is much more complicated, with the abundance of potentially conflicting filter residual test information plus a new type of failure detection test (the chi-squared test). The DIINS inertial error model is also much more complex than the corresponding MINS DR (fdead reckoning) model. The fact that DIINS has two central "systems" in the INS's rather than the single DR "system" in MINS introduces an additional level of reconfiguration.

The potential for DIINS to eventually be modified to meet the future submarine requirement also calls for an increased attention to the issue of software reliability.

1.3 Summary Recommendations

After a detailed study of the requirements and careful analysis of the alternative options and systems the conclusions were the following:

1. The Ada Programming Language is the most suitable implementation language for the DIINS application,
2. The Telesoft Ada compiler is the best production quality compiler on the market,
3. A suitable real-time executive must be acquired to enhance the real-time capabilities of the Ada runtime system. It must be appropriate to the needs of the target configuration,
4. The Sun 3/60 (Sun 3/80) workstations provide the best price/performance capabilities as a host system for the DIINS application,
5. The Motorola 680X0 family of microprocessors are the most suitable target processors for the DIINS application, and

6. The VME bus architecture is a suitable bus architecture for the DIINS system based on its popularity, compatibility and performance.

It is therefore recommended that the Defence Research Establishment Ottawa acquire:

- three Sun 3/60 (Sun 3/80) workstations (including 800 megabyte hard disk, Sun O/S and accessories)

and

- RTAda Comprehensive Development System

One of the overriding considerations that swayed the decision in favour of the Sun 3/60 over the alternatives such as the Microvax II was cost. The Microvax II option would cost $80,000 which represented a sum of $20,000 more than the Sun 3/60 option. Another advantage that the Sun 3/60 workstation afforded over the Microvax II was in the area of performance. The Sun 3/60 was 2–3 times faster than the Microvax II for a fraction of the cost. Moreover, while the performance of the Microvax II would degrade as the number of users increased, the Sun 3/60 offered the option of enhancing overall processing power through the acquisition of additional workstations.

The Sun 3/60 option also offered advantages in the area of compatibility since, unlike the Microvax II, it is object code compatible with the target computer. Subsequent acquisition of a Sun 3/470 brings with it VME compatibility and hence a host of options and VME-based products for the target computer. The bus used with the Microvax II is the Q-bus which offers less possibilities than the VME including speed.

The choice of the ARTX package must carefully be examined and understood. It may seem at first glance to be a premature decision. After all ARTX presupposes that the target configuration is single processor. Despite this, its purchase was recommended based on the following considerations:

1. The cost of the host compiler from Telesoft would be the same as the cost of the host compiler and ARTX when obtained from Ready Systems. In short the acquisition of ARTX posed no additional cost but would result in potential savings.
2. If the decision were made to use multiple processors, the ARTX must be used together with another package called MPV,
3. ARTX (VRTX) was the only real-time executive that was integrated at the time with the Telesoft Compiler,
4. ARTX enhances the real-time capabilities of Ada runtime system with its deterministic approach to multitasking, and
5. ARTX was used in over 3000 real-time projects including PLANS and MINS, two successful projects completed by DREO. It is the most popular executive, controlling 75% of the market share.

Changing technology brings with it better products which could affect some of the decisions made during this study. Two of the areas of concern are the choice of the target processor and the choice of the real-time executive.

The Ada Software Engineering Institute has recommended that processors such as 68020 and 80386 with their complex addressing modes, large linear address space and 32-bit architecture are at present most suitable for use as target processors in an Ada application. This decision was only based on the optimization and maturity of compilers for these processors.

RISC technology promises more processing power than that currently attainable using CISC technology (Complex Instruction Set Computers, e.g. 68020, 80386 etc.). The Telesoft Corporation has taken more than four years to optimize its compilers to CISC based processors such as members of the 680X0 family.

A decision in the near future to use RISC processors should be based on the maturity of the compilers, not on MIPS, since an immature compiler can produce inefficient code structures and slower performance. The compiler may not take advantage of the RISC architecture.

While special purpose Digital Signal Processors such as TMS320 promise more speed and conformance to the architecture of the DIINS system than the 68020, it is not likely that Ada compilers would ever be targeted to them because it is not viable economically.

Since the conclusion of the study several companies have integrated their real-time executives with the Telegen 2 system, including MTOS (for multiprocessing) and pSOS. Since pSOS will be supported by Telesoft, the only advantage in replacing ARTX with pSOS is that support will be provided by Telesoft alone. There is little to gain by a flip-flop in this direction.

Also since the time of writing the MTOS a real-time executive, which is reputed to be better for multiprocessing, has been integrated with the Telegen 2 ADA compiler. While no comments as to the validity of these claims can be made this remains a non-issue until the DIINS simulations dictate whether the target processor is single processor or multiprocessor.

2 SOFTWARE REQUIREMENTS

The DIINS (Dual Inertial Integrated Navigation System) is a real-time, multi-sensor, optimally integrated navigation system that is currently being designed and developed by Defence Research Establishment Ottawa. It is expected to require the development of a fairly large (>20,000 lines) multi-task software package, to perform the real-time sensor data fusion from the six or seven different navigation sensors being integrated.

2.1 Large Embedded System

The system is characteristic of a real-time embedded system in that it is large and will be long-lived and will continually undergo change. It is therefore essential that the implementation language of choice support team development and be maintainable, understandable, efficient and reliable.

2.2 High-Level Language

Furthermore the DIINS system should be implemented in a high-level language rather than assembly language for ease of maintenance, productivity concerns and the availability of programming personnel for high-level languages.

2.3 Fault-Tolerant

A major requirement of the DIINS system is its ability to detect, isolate and recover from sensor failures. This suggests that the implementation language should have an exception handling mechanism which allows the development of fault-tolerant systems. The implementation language should also facilitate the dynamic reconfiguration of software based on the detection of impending failures. It is important that the programming language have mechanisms which would allow the reconfiguration to incur minimal or no overhead during runtime.

2.4 Rate of Input

The system should facilitate data input from the sensors at rates of 1 to 10 Hz. The system should account for data that is not provided within an allotted time.

2.5 Processing Time

The sensor data should be processed by the prefilter and Kalman filter tasks at an update time of $\leq$ 10 seconds (preferably about one second). As explained in section 1.2 above, this will likely require at least an order of magnitude more processing power than say a 68020/68881 running at 10 MHz.

2.6 Support for Concurrency

Within the DIINS system there will be three to five parallel threads of control, each representing a separate prefilter/filter task. Because of the parallel nature of the system, support for concurrency is vital both from an organizational and implementation point of view. This would ensure separate tasks are apportioned the responsibility for sensor input and computation and the capability exists for these activities to overlap.

2.7 Support for Data Abstraction

To reduce code size, maintenance and the programming effort for the DIINS system, it is important that the implementation language facilitate the definition of reusable code and data templates. These templates may be shared by the various parallel prefilter/filter tasks resulting in decreased programming effort, an elegant design and reduced maintenance costs.

2.8 Support for Real-Time

Due to the real-time nature of the system, support for interrupts, priority based scheduling of the prefilter, data collection and filter tasks as well as support for time (delays, time outs) is crucial.

The cyclical nature of the DIINS system also dictates support for time-slicing amongst the various tasks of the DIINS system, forcing a predefined order of execution. It is also required that the software support explicit tasking control to allow resumption, suspension or rescheduling of tasks within the DIINS system.

3 COMPARISON OF PROGRAMMING LANGUAGES

In this section we shall compare the Ada and the C languages based on the general software requirements of the DIINS system. This is not a comparison of the features of the two languages but rather a comparison to decide which language is more suitable as the implementation language for the DIINS system.

3.1 Large Embedded System

Most software engineering methodologies share a common view of the requirement for projects that involve several people and long times. It must be possible to:

1. Decompose the project into subtasks which can be developed independently,
2. Assemble the resultant components into an operational system, and
3. Manage the long-term maintenance and enhancement of the project.

While a tool such as a programming language cannot cope with all of these problems there are some aspects of a language that can significantly impact these issues.

In this section we shall compare the Ada and C programming languages in terms of their support for programming in the large — maintenance and decomposition.

3.1.1 Maintenance

The DIINS system is a research project that will be designed and redesigned many times before the system is completed.

One of the most important characteristics of a maintainable system is understandability. The person who is responsible for maintaining a system is usually not one of the original authors and even if so, the interactions are usually subtle, complex and easily forgotten.

One of the first problems of maintenance is to re-understand the existing code and quickly understand the implications of a change well enough to make the necessary changes. While one of the aspects of this for large systems is the ability to easily locate relevant information, there is also a counterbalancing need to hide information since human beings can only deal with small amounts of detail at one time.

It can be stated that readability is inversely proportional to compactness. While the C language is very compact and powerful it does so at the expense of readability. Many C programmers will argue that it is possible to write readable C programs. However expressions such as: Char(**X())[]; in C require a syntactician to discover their meaning. Moreover, since there is also a tendency to write the best possible code in a given language, this naturally works against writing readable code in C. C gains its efficiency from effective use of pointers, side effects, auto-increment and auto-decrement to produce code that is difficult to understand much less change. This results in significant effort and costs during the maintenance phase of the project.

The C programming language shows very limited support for information hiding, one of the effective weapons in designing maintainable systems. Consequently changes are much more difficult to localize and contain, resulting in larger maintenance costs.

The Ada programming language has much more readable syntax than C. It is much easier to locate relevant information, which is one of the aspects of maintaining a large system. At the same time Ada supports information hiding which confines the impact of a change from propagating to surrounding modules. Changes made to a data structure (if it is deemed private) in an Ada program may only require a recompilation of the program. This greatly simplifies the maintenance effort and significantly reduces costs.

3.1.2 System Decomposition

The Ada programming language supports the development of large systems by several people. It allows a system to be decomposed along natural lines in one of several dimensions — functional and data oriented.

Whatever the nature of the decomposition it is important that the resulting pieces should correspond to work assignments. In other words, each of the portions of the original project should be performed by an individual in isolation.

The Ada programming language accomplishes this by enabling various subsystems such as data collection tasks and prefilter tasks to be operated on by individual project team members. Its support for separate compilation ensures that there is consistency across compilation boundaries. In short it ensures that there is consistency in the work done by individual team members, while at the same time affording them the independence of working in isolation. C on the other hand only supports independent compilation. While modules can be physically compiled separately, there is no type checking across the compilation boundaries. This makes it much more difficult to perform system decomposition and to assemble the pieces into an operational system, since there is not nearly the same level of checking as that afforded by Ada.

C does not support data abstraction and hence a system can only be decomposed using functional decomposition. With its use of abstractions, Ada allows common data structures for the prefilter tasks to be factored out thereby making system decomposition much easier and reduces the programming effort.

3.2 Fault-Tolerant

The Ada programming language also aids in the production of fault-tolerant software by providing an exception handling mechanism. In languages such as C a user must test for the occurrence of an error and direct the control flow to an error routine where the error is handled. Since it is impossible to decide beforehand all the things that can go wrong in a program it is very difficult to develop a fault-tolerant system using C.

Moreover, the C programming language does a lot to discourage the development of fault-tolerant systems. It does not ensure that array bounds are not exceeded, that values passed to a

subroutine agree in type or that garbage is not returned in a function call. While these errors are easy to locate in a small system, they become exaggerated as the size of the project increases.

Within Ada the error seeks the error routine. No explicit control is required to direct traffic of errors to particular error routines. Ada accomplishes this by latching into the interrupt handling mechanism of the language. The exception handling mechanism can be easily tailored using Ada to meet the needs of the application thereby enabling the realization of fault-tolerant systems.

3.3 Support for Concurrency

The Ada programming language has built-in support for multitasking. Using tasks and the rendezvous a programmer can easily create a variety of multitasking primitives such as semaphores, event flags and message queries. Inter-task communication is achieved via the rendezvous mechanism and by shared variable support.

The C programming language does not directly support multitasking. It considers multitasking to be in the domain of the operating system. However its low level approach to things is such that suitable primitives could easily be written.

3.4 Support for Data Abstraction

The C programming language shows no support for data abstraction. It does not allow for definition of reusable code and data templates (generics). This results in an increase in the programming effort and an increase in the maintenance effort since there is more code to maintain. Ada easily allows the definition of reusable code and data templates. This reduces the programming effort and results in a system that is more elegant and maintainable.

3.5 Support for Real-Time

The Ada programming language has limited support for real-time applications while the C programming language provides almost no support. While Ada supports a priority based scheduling scheme, interrupt handling and coding of time sensitive logic involving delays or time-outs, it has many limitations in the domain of hard real-time applications. It lacks support for time-slicing and explicit tasking control. However all of these problems can be solved using an appropriate Real-time executive which would extend the capabilities of the Ada runtime system. This is the approach employed by the C programming language anyway which achieves real-time capabilities through calls to the operating system.

3.6 Programmer Availability

Programmers skilled in the ADA language are still far fewer than those skilled in the C language. However the DIINS development project size and schedule are such that only a few such programmers will be required, and adequate numbers of ADA programmers have been identified as available. This programmer availability situation is also expected to improve with time.

4 PROGRAMMING SUPPORT ENVIRONMENT

Development of real-time embedded systems differs from other software development in that it requires the use of two computers rather than one; the **host** system on which the application is developed and the embedded computer, which is referred to as the **target**. The target computer has limited functions and limited memory. Therefore it is necessary to provide an integrated development environment of tools and utilities common to both host and target. This integrated development environment is termed the Programming Support Environment.

The selection of an appropriate Programming Support Environment ensures compatibility between host and target systems and reduced development time. To facilitate development on the host systems the following tools would be required on the host:

1. Validated Ada compiler,
2. Pretty printer,
3. Library manager,
4. Host performance profiler,
5. Source level debugger,
6. Make tool, and
7. Global optimizer.

These tools are often supplied along with the compiler.

4.1 Compiler

At present there are over 145 base and 58 derived compilers making the choice of an Ada compiler non-trivial. All compilers are validated. However validation does not mean that the compilers are:

1. Efficient:
    a. fast object code,
    b. small object code,
    c. small compiler size, and
    d. fast compilation speed.
2. Adequately supported by the vendor, or
3. Tested on real (or similar) projects.

Since the majority of Ada compilers have focused on validation rather than real-time implementation, only a mere handful of present compilers fulfill the requirements of the DIINS system. The host and cross compiler also must be compatible with each other . The target processor must also be one of the more popular processors such as 68020 or 80386.

After a detailed evaluation, the list of eligible Ada compilers was reduced to the Verdix Ada Development System (VADS) and Telegen 2, whose real-time implementations could be

targeted to either the Intel 80386 or the Motorola 68020. Both of these compilers produce fast efficient object code as exemplified by their usage in many real-time projects.

The VADS system from Verdix has a more user friendly environment for the host system and includes a debugger which is superior functionally to that of Telegen 2. However, the Telesoft compiler (Telegen II) showed 10% faster execution speed and slightly more efficient object code. Figures 1 and 2 show the significant improvements made by Telesoft in terms of execution speed and efficiency over several versions of their compiler.

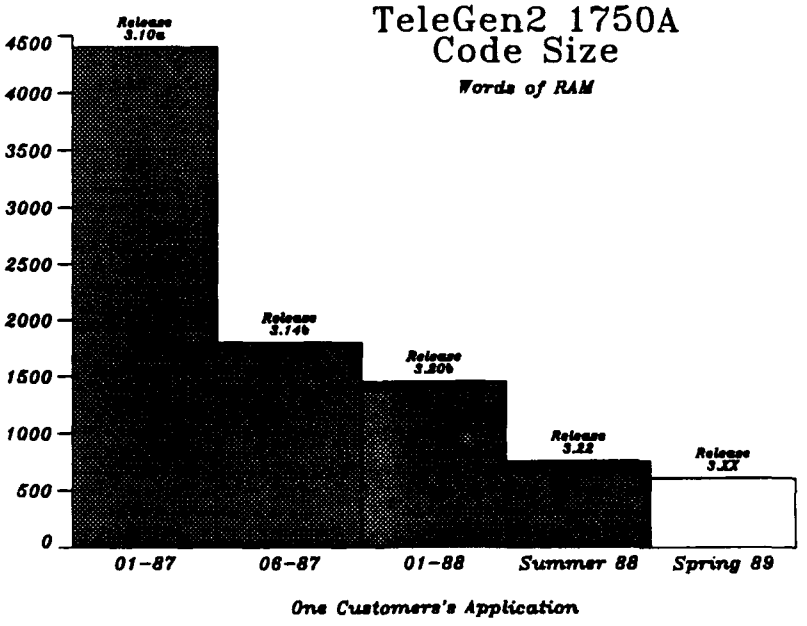


Figure 1 Telegen2 Code Size

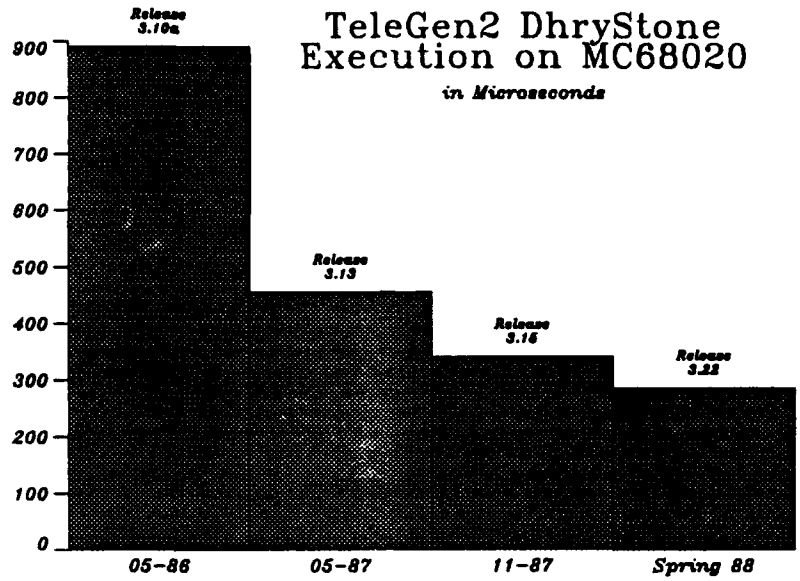


Figure 2 Telegen2 Execution Speed on a Motorola 68020

The runtime system is also significant in the choice of a compiler for real-time applications. The runtime system of both the Telesoft and Verdix compilers is inappropriate to the needs of a real-time system. For example performance degrades as the number of tasks increases.

This situation can also be alleviated by choosing an appropriate runtime executive, which enhances the real-time capabilities of the Ada runtime system. The Telesoft compiler can be successfully integrated with a majority of the runtime executives on the market currently utilized in hard real-time applications. The list includes ARTX (one of the more popular executives), MTOS (for multiprocessing) and pSOS. Although the Verdix corporation is in the process of negotiating these alliances, the realization is several months away.

Finally, a study conducted for NATO at John Hopkins University, has shown that the Telesoft Compiler in most appropriate in the navigation applications. As a result, the Telegen 2 system was chosen for the DIINS application.

4.2 Global Optimizer

The global optimizer, a necessary component of the compiler, is required to make compiled programs more efficient. In the majority of cases the code is both smaller and faster than when compiled without the optimizer.

The global optimizer supplied with Telegen 2 performs a variety of optimizations such as the "inlining" of subroutines thereby saving the overhead of procedure calls and removing code that is not called.

4.3 Library Manager

Tools are required in order to maintain control over the DIINS program library. These enable manipulation of programs in the library, facilitate copying of programs between sublibraries, and allow version control and compression of dead code. This improves productivity since the programmer does not have to create utilities or remember operating system commands to accomplish these functions.

4.4 Source Level Debugger

The Programming Support Environment should contain a symbolic debugger which enables programmers to establish an interactive debugging session by working the application from the debugger. The debugger should have capabilities such as:

1. Setting of breakpoints,
2. Display of variable names,
3. Display of task status information,
4. Trapping of unhandled exceptions,
5. Log files which contain user requests and debugger responses for later program analysis, and
6. Machine level access to enable the experienced programer to examine the actual contents of device systems.

The Telegen 2 supplies a source level debugger which accomplishes these functions.

4.5 Host Profiler

Perhaps the most useful tool in a Programming Support Environment for a real-time application is a profiler. It is said that the performance of a large system is often dictated by small portions of it. Profilers are required in order to improve efficiency and monitor execution of the DIINS system host and target. These profilers pinpoint possible inefficiency in a compiled Ada application and illustrate where program alteration might improve performance. A host profiler is supplied as part of the Telegen 2 system.

4.6 Downloader and Receiver

In an embedded system there is always the need to have a tool to download executable format load modules to the target processor and to upload the contents to the host system from the target computer. A Downloader and Receiver is part of the Telegen 2 configuration. This facilitates the downloading of Motorola 68020 S-Records to the target computer.

5 HOST SYSTEMS

After selecting the Telegen 2 Development System the consultants embarked on a study to select an appropriate host system on which the DIINS application could be developed. At the time of the study, the only possible choices for a host system for the Telesoft Ada compiler were the Vax family of computers and the Sun family of workstations. Due to the prohibitive cost of the high end members of the Vax family the choice was limited to the Microvax II or the Sun 3/60 family of workstations.

5.1 Costs

The cost of the Microvax II hardware alone was almost equal to that of the entire system, both hardware and software, of the Sun 3/60 option. Moreover, due to its multi-user license the cost of all software components on the Microvax II were much higher than that for the Sun 3/60. While it is true that the cost of the software license for the Sun 3/60 increases with the number of users, options such as remote login capability pushed the decision in favour of the Sun 3/60.

If the Microvax II option were selected, it would require a one-time expenditure in excess of $80,000 for both hardware and software. For the same cash outlay, the equivalent of three workstations together with software could be acquired. Each workstation has 2–3 times the processing capability of the Microvax II. If the cost for the total number of Sun 3/60 workstations was too high, then one or two could be secured initially, thereby satisfying the budget constraints.

5.2 Performance

The Sun 3/60 family of workstations has two to three times the speed of a Microvax II. The performance of the Microvax II degrades as the number of users increases. Usually this saturates the processing power of the machine and results in very poor response times and poor productivity. If the Sun 3/60 were chosen as the host, additional workstations could be acquired as the project team enlarged, resulting in greater productivity. Also with the acquisition of more workstations, DREO could be assured that the computer becomes more powerful rather than less powerful as in the case of Microvax II.

5.3 Compatibility with Target

The Sun 3/60 workstation is 68020 based which makes it compatible with the Motorola family of processors. With the host 68020 based and the target 68020 based, potential for translation or compatibility problems are eliminated, a definite advantage over the non-68020 based Microvax II. Incompatibilities occur between the host and cross compiler in the Microvax II system because information is stored on the host in the opposite order to that of the 68020 (big endian versus little endian).

5.4 User Friendly

The Sun 3/60 uses windowing software which enables several tasks to be operated concurrently. It allows a user to view several Ada files at the same time. This is extremely useful in an environment, such as DIINS, where a programmer uses components from several libraries and may need to work with them at the same time. The programmer could also be editing another task in another window resulting in greater programmer productivity.

5.5 VME

The Sun 3/60 does not have a VME backplane. Therefore DREO could subsequently acquire a Sun 3/470 workstation which has a VME backplane. The VME is an industry standard. However the Microvax II is not compatible with VME. It uses its own Dec-based bus called Q-bus which is also much slower than the VME.

6 TARGET SYSTEMS

6.1 Microprocessors

In this section we shall compare the suitability of a variety of microprocessors as possible candidates for the target computer. This is not a technical comparison of the capabilities of the microprocessor but rather a comparison in the context of the DIINS application.

6.1.1 iAPX80386

The Intel 80386 processor complemented by its use of the 80387 floating point processor is a very suitable processor for the DIINS application.

It is characterized by a large linear address space (4 gigabyte segment), and complex addressing modes which map well into the high-level capabilities of Ada.

In addition, a concurrent Ada application causes many context switches when switching from one task to the next. This results in large delays since the processor must save all the registers and task states before switching to another task. The 80386 does this in hardware with a single instruction making it a good choice for an Ada application.

However much more than the microprocessor architecture must be taken into account before choosing a target microprocessor. One of these considerations is how mature is the Ada compiler and does it take advantage of the capabilities of the microprocessor. Another consideration is the mapping of the architecture of the application onto the capabilities of the microprocessor.

Intel has long kept away from Ada. It was only in 1989 that it negotiated an agreement with Telesoft to target the compiler to its processors. Over 90% of the Ada compilers are targeted to the Motorola processors. Over the past four years Telesoft and other companies have spent considerable effort to optimize their compilers to the Motorola microprocessors. As a result of Intel being four years late, the race may be lost.

6.1.2 iAPX80960

The Intel 80960 is a new chip which offers as much as 66 MIPS. It uses a combination of RISC and CISC technology. Unlike the traditional CISC microprocessors the iAPX80960 does not require a floating point processor since it has these capabilities built in. Like the 80386, it has a large address space, complex addressing modes and fast context switching time. The 80960 processor comes in 3 architectures — core, numerics and protected which are supersets of the preceding architecture. The numerics architecture is used in computation-intensive applications, while the protected architecture is used in CAD/CAM and robotics applications.

At present Tartan Laboratories, a company in Pennsylvania, has targeted its compiler to the 80960 with good results. However, the microprocessor is too new to allow the compiler developers to take advantage of its capabilities.

6.1.3 TMS320

Special purpose Digital Signal Processors such as the TMS320 are very suitable for the Kalman filtering phase of the DIINS application. Its ability to multiply vectors by matrices with great speed (60 nanoseconds) would definitely be very useful for the DIINS application. It employs a great deal of parallelism in its ability to perform calculations in parallel as well as parallelism in its internal bus architecture. However it is unlikely that an Ada compiler will ever be developed for the TMS320. Its specialization and limited usage pose considerable risk to any company wishing to develop an Ada compiler targeted to it.

6.1.4 MC680x0

The Motorola family of microprocessors is currently the most ideal microprocessor for the DIINS application. While it is not technically superior to the TMS320 or iAPX80386, other considerations motivate this decision.

During the phases of compiler development, compiler companies focus on several phases or activities in optimizing a compiler for a particular microprocessor. Thus far Telesoft has succeeded in optimizing its code generation to the Motorola 680X0 family. Its compiler takes advantage of the newer instructions introduced within the Motorola 68020 family such as the newer bit manipulation instructions, and the long word multiply and divide. The next phase of the development of the Telesoft compiler will see a much more deterministic approach to multitasking thereby reducing the responsibilities of the real-time executives. The Motorola 680X0 family has a large linear address space, complex addressing and 32-bit architecture.

It was deemed one of the most suitable target processors by the Ada Software Engineering Institute. The RISC microprocessors such as Motorola 88000 are several years behind the 680X0 in terms of Ada compiler maturity.

Finally, the great number of vendors currently developing VME based products motivate the decision — the 680X0 family of microprocessors is most suitable target processor for the DIINS application.

With the announcement of the arrival of the Motorola 68040 further study is required in this area.

6.2 Real-Time Executives

6.2.1 Problems of Ada Runtime System

The Ada programming language has a built-in support for real-time and concurrent applications. Its support for concurrency is exemplified through the presence of multitasking primitives and the ability of Ada tasks to communicate with each other via shared variables or the rendezvous. Real-time support is established through its support for Interrupts and its ability to manipulate time via the calendar package. All of the above facilities are provided through the Ada runtime system.

However, the majority of runtime systems used by the compilers employ a non-deterministic non real-time approach to multitasking. To ensure proper implementation of the DIINS system, it is essential that the runtime system provide the necessary target performance and real-time functionality. The Ada runtime system lacks some facilities thereby limiting the scope of its applications.

In this section we will explore some of the limitations of the Ada runtime system and describe how some of these limitations can be overcome through the use of an appropriate runtime executive.

Interrupt Handling Interrupts, by definition, are asynchronous and cannot wait to be serviced. Within Ada the mechanism used to handle interrupts is the rendezvous, which is synchronous and does wait. The queuing mechanism used in the Ada tasking model is inappropriate for certain kinds of interrupts which could be lost. Another problem associated with the handling of interrupts is most conventional Ada runtime systems have a high interrupt latency (time of arrival of signal to execution of first instruction in the Interrupt Service Routine). Interrupts are also disabled upon entry to the runtime system. It is important that this time be minimized.

Determinism One of the requirements of most real-time applications is the ability to perform the appropriate action within severe time constraints. In short, "deterministic behaviour" is essential for real-time systems.

Call to Ada tasks are handled in a non-deterministic manner, so one cannot be assured that the action required of a task could be completed within the desired constraints. Furthermore, most Ada runtime systems provide degraded performance as the number of tasks in the system increases. The delay statement is another reason for concern. The Language Reference Manual states the time allotted in a delay statement is the minimum time of the delay rather than the actual time. These limitations pose severe problems in the implementation of real-time applications. In DIINS time-sensitive actions may not be completed on time, due to the unpredictable characteristics of the Ada runtime system.

Rendezvous and Multitasking Primitives In the design of real-time systems, considerable overlap is required in computation and sensor activity. To achieve the necessary overlap, multitasking primitives such as semaphores, message queues and event handling mechanisms are required. These primitives can be built using Ada tasks and the rendezvous mechanism. However they perform poorly because of excess overhead.

Scheduling In the realization of real-time systems some consideration must be given as to how and when a task is rescheduled. It is always desirable to run the highest priority task that is ready, at any point in a program. Within an Ada program, however, rescheduling is only permitted at certain "synchronization" statements which are few. In addition Ada tasks are scheduled in a FIFO manner and not on the basis of priority. There is no mechanism to preempt a task or resume a task, which is often required in a real-time application.

Support for time-slicing is essential for tasks of equal priority because of the cyclic nature of the DIINS system. The Ada runtime system supports preemptive scheduling in favour of task time-slicing.

Asynchronous Task Communication Ada does not provide primitives for asynchronous task communication. All inter-task communication is provided through the rendezvous (and shared variables). The rendezvous is synchronous which makes it difficult and inefficient to implement entry calls without wait.

6.2.2 Ada Real-Time Executive (ARTX)

The ARTX is a real-time executive (or operating system) which helps overcome the limitations of the Ada runtime system by enabling the Ada application to make calls to the ARTX operating system. These actions are often performed much faster than by using the facilities of the Ada real-time system. While this subtracts from some of the advantages of Ada such as portability, it enhances the real-time support provided to Ada in terms of functionality and performance.

Time-slicing The ARTX allows a task to be given a quantum of time to execute. This time may be adjusted dynamically on a per task basis. No task is permitted to execute longer than its time-slice. This ensures proper implementation of the DIINS system to ensure that prefilter and data collection tasks get their fair and appropriate share of processing time.

Interrupt Handling The ARTX facilitates the handling of interrupts using interrupt procedures (called by the operating system) rather than tasks. This ensures that interrupt signals are not lost. The interrupt latency time is in the order of 12 microseconds, which is quite acceptable for most real-time applications.

Determinism The Ada Real-Time Executive (ARTX) provides a deterministic approach to multitasking. Ceiling for delays, latency times, context switching and rendezvous times are available for scrutiny. Hence the designer of an application is not at the mercy of the Ada runtime. Some of the published values are listed in the table below.

Delay Preemption	Yes
Preemption for Interrupts	Yes
Fixed Overhead	Yes
Interrupts Disabled	$< 12\mu$seconds
Context Switch	40 μseconds

Runtime Latency	60 μseconds
Rendezvous Times	120 μseconds

(All numerical values for a Motorola 68020, 20MHz, 1 wait-state.)

Rendezvous and Multitasking Primitives Ada allows multitasking primitives to be built using rendezvous and Ada tasks. This results in a large overhead. The ARTX system allows the definition of mailboxes, semaphores, queues and event flags with significantly less overhead than that imposed by the Ada runtime system.

Scheduling ARTX allows explicit control of the scheduler by providing a package, called Scheduler, which facilitates the enabling and disabling of task scheduling. This would allow a section of code to be executed to completion. The only place where code is guaranteed to execute to completion in Ada is during the rendezvous. The ARTX also provides facilities to suspend and resume a task which are missing in Ada.

Asynchronous Task Communication The ARTX offers asynchronous communication in addition to the synchronous communication offered by Ada, which allows entry calls without wait. This is particularly useful in the DIINS system where the data collection tasks would therefore not need to synchronize with prefilter tasks to exchange data (thus avoiding the potential loss of some sensor data).

6.3 Bus Architectures

A significant factor in the choice of board level components of a target system is the choice of the system bus over which commands and data are transmitted. This choice is more significant for a real-time application, such as the DIINS system, where fast data transfer and signalling is crucial to its reliability. The impact of bus selection figures strongly not only in system performance, but also in the engineering time of configuring and enhancing the system in the future.

Several popular buses are available for comparison. They include:

1. Multibus II (Intel)
2. VME (Motorola, Mostek, Phillips)
3. Q-bus (Dec)
4. STD (IEEE P961)
5. S-100 (IEEE 696)
6. Versabus (Motorola)
7. Nubus (Texas Instruments)
8. GPIB (IEEE 488)

The Q-bus was dropped from further consideration since the choice of the host was not Dec-based. The STD and S-100 buses which are first generation buses were also dropped because they connect older processors and are unreliable (susceptible to corrosion and shorting). The Versabus and Nubus were eliminated from the list because they are not widely used so components are not readily available. The GPIB (general purpose instrumentation bus) is old technology and cannot connect memory to CPU because it is not a direct access bus (needs an interface at each end with clumsy software protocol).

This left the Multibus II and VME system as the only candidates. In this section we shall compare these two buses based on technical and business factors.

6.3.1 Synchronous Versus Asynchronous

The VME system is an asynchronous bus structure and Multibus II is synchronous. The advantages and drawbacks of each design must be examined and evaluated to determine which is the better solution for the DIINS system.

The goal of the VME system is to maximize the performance of a multiprocessor system. An asynchronous bus does not have a defined speed of operation. Thus by choosing an asynchronous bus structure, the transfer rate of the bus can adjust to the transfer rate of the devices using the bus. If the target system for the DIINS system consisted of several processors, the asynchronous nature of the VME system could allow data to be transferred by each processor at its fastest possible rate.

In contrast the Multibus II defines its bus, the IPSB, to operate at any frequency to a maximum of 10MHz.

6.3.2 Data/Address Bus

The VME and IPSB (of Multibus II) each have 32 data transfer lines. Thus when a master takes control of the bus, the size of the data can be specified on a cycle-by-cycle basis. This allows 8, 16 or 32 bit processors on the system talking to 8, 16 or 32 bit memory or I/O. However, the IPSB has the additional capability of transferring 24 bit data due to the capabilities of the iAPX 386 processor.

The VME system gives the option of choosing 16, 24 or 32 bits of address interface while with the IPSB only 16 or 32 bits of address may be sent down the line. The address and data lines are multiplexed onto the same pins in the Multibus II, while on the VME different pins are used.

6.3.3 Arbitration Scheme

The bus arbitration scheme is an important characteristic of a multiprocessing system. It determines which of the different masters in a system gains control of the bus at any one point in time.

The VME bus has one global Arbiter which is responsible for control of the bus. It does so through its own on-board requester. The IPSB uses a distributed arbitration scheme. Each board in the IPSB system attempts to place its arbitration ID number on the bus. If the number

on the bus matches the arbitration ID number, that master becomes the bus owner. The IPSB has only one bus request line shared by all the masters in the system, and thus is rigidly defined to work one way. The VME system on the other hand, has four prioritized bus request lines with more than one master sharing a line. The VME system offers a wider variety of options by allowing three different choices for arbitration and two different choices for requesters. Thus the VME system would allow wider flexibility in configuring the system to the need of the DIINS application.

6.3.4 Interrupt Handling

This, perhaps, is the area where the VME and Multibus II differ the most. The VME system specifies dedicated hardware for handling and generating interrupts, relieving other masters of this task. On the other hand, the Multibus II system handles interrupts without dedicated devices. Their solution is accomplished by passing a message from one master to another, which requires interrupting devices to first acquire control of the bus and then send the information to the interruptee. The VME proposal handles interrupts faster than the Multibus II by using the hardware solution.

6.3.5 Processor Independence

The VME and Multibus II can be used with a variety of processors from Motorola and Intel respectively. However, Intel currently supports the VME bus while Motorola processors are not currently used with Multibus II. This gives the VME bus greater processor independence. Should technological advancement offer faster and suitable processors, DREO's investment in the VME bus would not be lost.

6.3.6 Multiple Vendors

Since no one vendor can serve the entire needs of a marketplace, it is important that board components for a given bus be available from a variety of vendors. Both VME and Multibus II have a wide variety of products available for their buses from Dy-4, Plessey, Motorola etc. This ensures that there is a ready availability of suitable bus products and ensures competitive pricing. The VME system has a larger market share and thus a slight edge in this category.

7 RESULTS

As a result of this requirements study and in line with the recommendations summarized in section 1.3 above, the following items were purchased to support the design and simulation analysis of DIINS:

1. two Sun 3/60 workstations with 4 MByte RAM each (one 19" monochrome and one 16" colour monitor),
2. 4 MByte RAM upgrade for one workstation (the server),
3. one 327 MByte hard disk drive,
4. one 670 MByte hard disk drive,
5. one 1/4 inch cassette tape drive (60 MByte),
6. one NEC LC890 laserprinter,
7. the SUN OS 3.5 operating system,
8. Ready Systems RTAda Comprehensive Development System software,
9. Precision Visuals PV Wave plotting and data analysis software and
10. ArborText Inc. "The Publisher" documentation software.

This SUN network was also connected by ethernet to the Electronics Division VAX 3600, which contains all of the MINS development environment software (as well as the development environment software for other integrated navigation systems such as PLANS etc.). A considerable amount of this previously developed software could be used for DIINS development.

Shortly thereafter, when additional funds were available, this network was enlarged by the purchase of:

1. expanded RAM to bring the two 3/60 stations to 16 MBytes,
2. one Sun 3/80 workstation, with 16 MByte RAM and a 19" mono monitor,
3. with a 3.5" floppy drive and
4. software for a DOS window.

This Sun 3/80 then became the file server for the other two workstations. A PC was also added to the network (33 MHz 80386), but it was not primarily for DIINS work.

REFERENCES

[1] J.C. McMillan, "Multisensor Integration Techniques in the DREO Marine Integrated Navigation System (MINS)", DREO Report 986, August 1988.

DOCUMENT CONTROL DATA

(Security classification of title, body of abstract and indexing annotation must be entered when the overall document is classified)

1. ORIGINATOR (the name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Establishment sponsoring a contractor's report, or tasking agency, are entered in section 8.)

DEFENCE RESEARCH ESTABLISHMENT OTTAWA
OTTAWA, ONTARIO
K1A 0Z4

2. SECURITY CLASSIFICATION (overall security classification of the document including special warning terms if applicable)

UNCLASSIFIED

3. TITLE (the complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S,C or U) in parentheses after the title.)

DEVELOPMENT ENVIRONMENT FOR DIINS (DUAL INERTIAL INTEGRATED NAVIGATION SYSTEM) (U)

4. AUTHORS (Last name, first name, middle initial)

MCMILLAN, J. CHRIS and RAMOTAUR, Rudy

5. DATE OF PUBLICATION (month and year of publication of document)	6a. NO. OF PAGES (total containing information. Include Annexes, Appendices, etc.)	6b. NO. OF REFS (total cited in document)
MAY 1991	32	1

7. DESCRIPTIVE NOTES (the category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.)

DREO TECHNICAL NOTE

8. SPONSORING ACTIVITY (the name of the department project office or laboratory sponsoring the research and development. Include the address.)

DEFENCE RESEARCH ESTABLISHMENT OTTAWA
OTTAWA, ONTARIO K1A 0Z4

9a. PROJECT OR GRANT NO. (if appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant)	9b. CONTRACT NO. (if appropriate, the applicable number under which the document was written)
PROJECT 041LJ	

10a. ORIGINATOR'S DOCUMENT NUMBER (the official document number by which the document is identified by the originating activity. This number must be unique to this document.)	10b. OTHER DOCUMENT NOS. (Any other numbers which may be assigned this document either by the originator or by the sponsor)
DREO TECHNICAL NOTE 91-5	

11. DOCUMENT AVAILABILITY (any limitations on further dissemination of the document, other than those imposed by security classification)

(X) *Unlimited distribution*
( ) Distribution limited to defence departments and defence contractors; further distribution only as approved
( ) Distribution limited to defence departments and Canadian defence contractors; further distribution only as approved
( ) Distribution limited to government departments and agencies; further distribution only as approved
( ) Distribution limited to defence departments; further distribution only as approved
( ) Other (please specify):

12. DOCUMENT ANNOUNCEMENT (any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). However, where further distribution (beyond the audience specified in 11) is possible, a wider announcement audience may be selected.)

UNCLASSIFIED

SECURITY CLASSIFICATION OF FORM

13. ABSTRACT (a brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), or (U). It is not necessary to include here abstracts in both offical languages unless the text is bilingual).

With a view to satisfying the navigational requirements of TRUMP, CPF and future submarines, DREO initiated an investigation into the possible extension of MINS (Marine Integrated Navigation System) to accommodate the two inertial navigation systems which each of these platforms were expected to have, in addition to the various navigation sensors that MINS already integrates (GPS, Transit, Loran-C, Omega, Speedlog and Gyrocompass). It soon became clear that the ideal solution would be a new system with a different architecture than MINS. The new concept was called DIINS (Dual Inertial Integrated Navigation System). A study was therefore initiated to determine the most appropriate hardware and software to be used in the design and development of DIINS. This report contains a brief description of the recommended hardware and software solution, as well as the rationale for their selection.

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus, e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus-identified. If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

NAVIGATION

MINS

DEVELOPMENT ENVIRONMENT

UNCLASSIFIED

SECURITY CLASSIFICATION OF FORM